



# *Geant4: Physics, Models and Cuts*

---

*Alexei Sytov*

*INFN – INFN Ferrara*

*A lot of material by G.A.P. Cirrone, L. Pandola, J. Pipek*

*Geant4 Course*

*XXII Seminar on software for nuclear, subnuclear and applied  
physics*

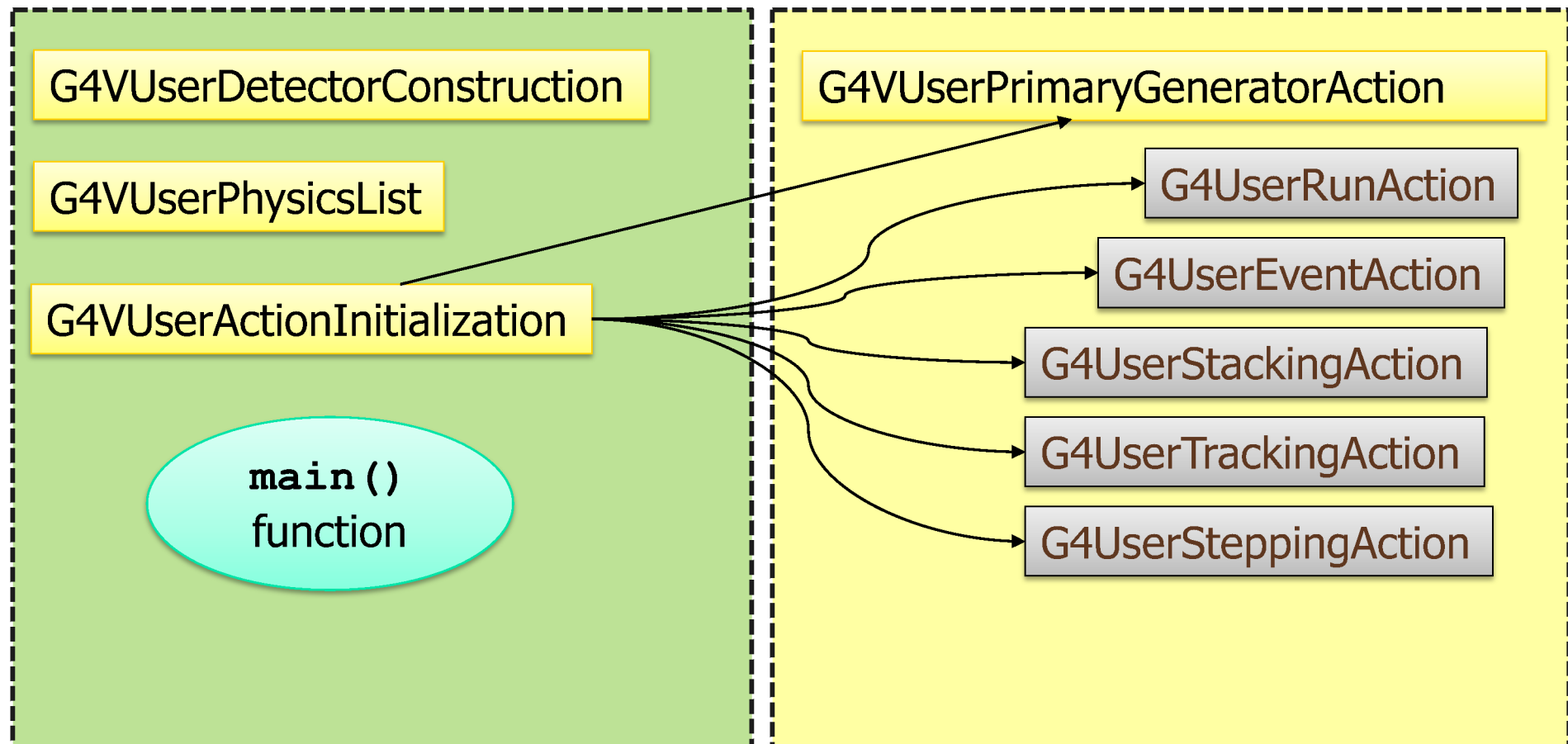
*Alghero, June 8<sup>th</sup>- 13<sup>th</sup>, 2024*

# Mandatory (and optional) user classes



## At initialization

## At execution

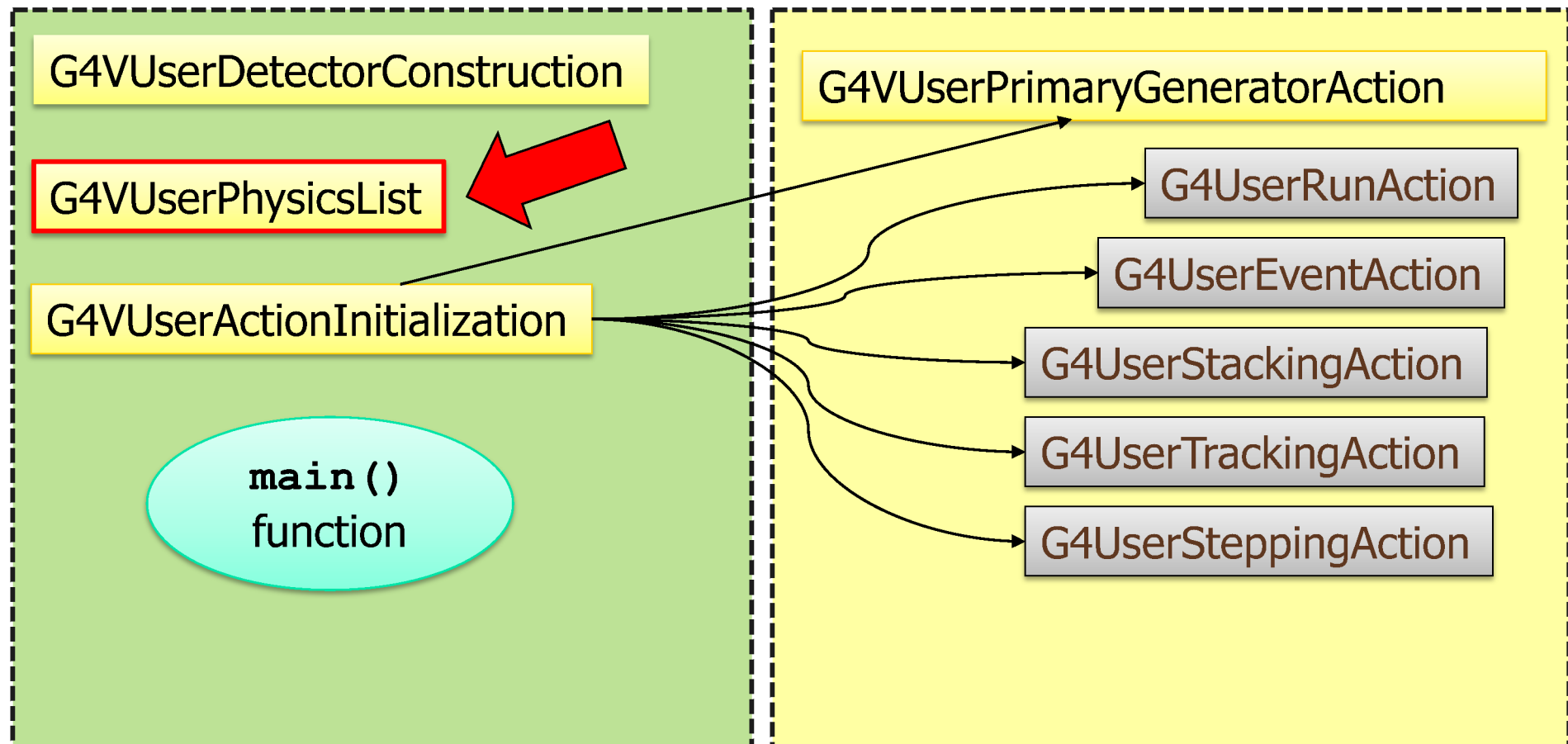


# Mandatory (and optional) user classes



## At initialization

## At execution

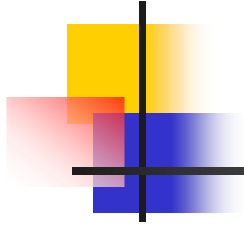




# Outlook

---

- Physics in Geant4 – motivation
- Particles & processes
- Physics lists
- Production cuts
- Electromagnetic/hadronic physics



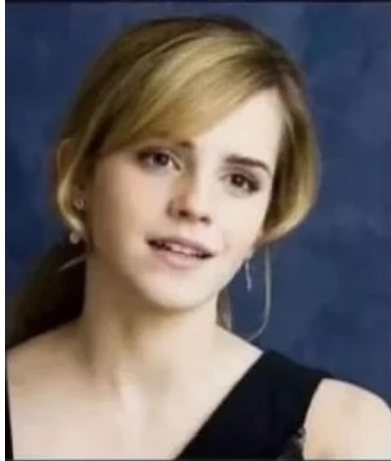
“Shouldn’t there be just one  
universal and complete physics  
description?”

**No.**



## **Rich Guy -**

Money doesn't matter.



## **Beautiful lady -**

Beauty doesn't matter.



## **Physicists -**

Friction, air resistance, shape of mass, mass of rope, mass of pulley doesn't matter



# Physics – the challenge

---

- Huge amount of **different processes** for various purposes (*only a handful relevant*)
- **Competing descriptions** of the **same** physics phenomena (*necessary to choose*)
  - fundamentally different **approaches**
  - balance between **speed** and **precision**
  - different **parameterizations**
- Hypothetical processes & exotic physics

**Solution:** Atomistic approach with modular **physics lists**



# Part I: Particles and Processes

---





# Particles: basic concepts

---

- There are three levels of class to describe particles in Geant4:
- **G4ParticleDefinition**
  - Particle **static properties**: name, mass, spin, PDG number, etc.
- **G4DynamicParticle**
  - Particle **dynamic state**: energy, momentum, polarization, etc.
- **G4Track**
  - Information for tracking **in a detector simulation**: position, step, current volume, track ID, parent ID, etc.



# Particles in Geant4

---

- Particle Data Group (PDG) particles
- Optical photons (different from gammas!)
- Special particles: geantino and charged geantino
  - Only transported in the geometry (no interactions)
  - Charged geantino also feels the EM fields
- Short-lived particles ( $\tau < 10^{-14}$  s) are not transported by Geant4 (*decay applied*)
- Light ions (as deuterons, tritons, alphas)
- Heavier ions represented by a single class: G4Ions

Particle name	Class name	Name (in GPS...)	PDG
electron	G4Electron	e-	11
positron	G4Positron	e+	-11
muon +/-	G4MuonPlus G4MuonMinus	mu+ mu-	-13 13
tauon +/-	G4TauPlus G4TauMinus	tau+ tau-	-15 15
electron (anti)neutrino	G4NeutrinoE G4AntiNeutrinoE	nu_e anti_nu_e	12 -12
muon (anti)neutrino	G4NeutrinoMu G4AntiNeutrinoMu	nu_mu anti_nu_mu	14 -14
tau (anti)neutrino	G4NeutrinoTau G4AntiNeutrinoTau	nu_tau anti_nu_tau	16 -16
photon ( $\gamma$ , X)	G4Gamma	gamma	22
photon (optical)	G4OpticalPhoton	opticalphoton	(0)
geantino	G4Geantino	geantino	(0)
charged geantino	G4ChargedGeantino	chargedgeantino	(0)



# Processes

---

How do particles interact with materials?

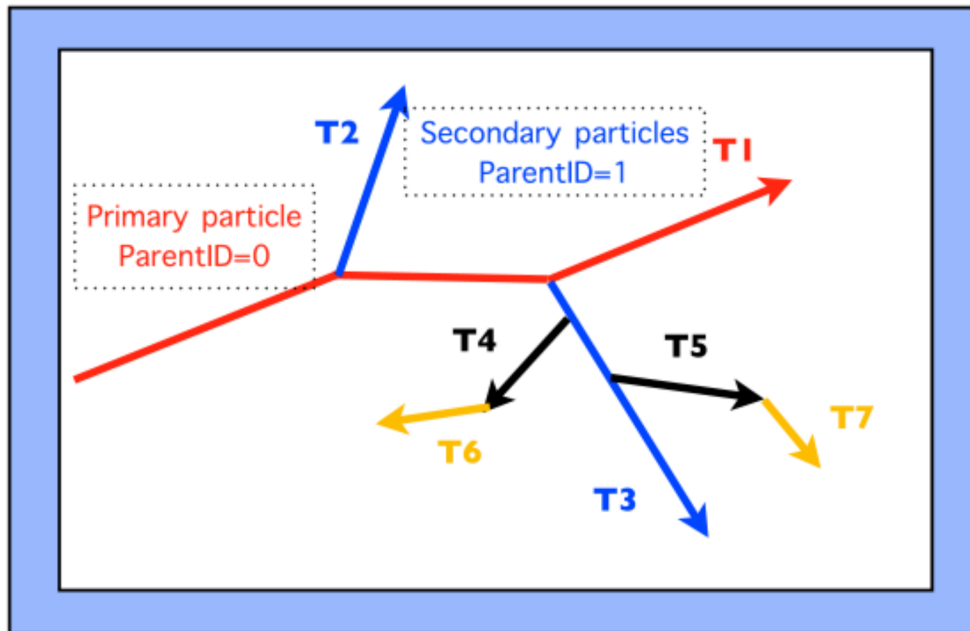
- Responsibilities:
  - decide **when** and **where** an **interaction** occurs
    - GetPhysicalInteractionLength...() → **limit the step**
    - this requires a **cross section**
    - for the transportation process, the **distance** to the **nearest object**
  - **generate** the **final state** of the interaction
    - changes **momentum**, generates **secondaries**, etc.
    - method: DoIt...()
    - this requires **a model of the physics**



## Part II: Tracking and cuts

---

# Geant4 way of tracking



- **Force step** at geometry **boundaries**
- All **AlongStep** processes **co-work**, the **PostStep** **complete** (= only **one** selected)
- Call AtRest actions for particles at rest

- Secondaries saved at the top of the stack: tracking order follows 'last in first out' rule:

T1 → T3 → T5 → T7 → T4 → T6 → T2

# Tracking verbosity

UI command: `/tracking/verbose 1`

Primary  $\gamma$

```
*****
* G4Track Information:  Particle = gamma,  Track ID = 1,  Parent ID = 0
*****

Step#    X(mm)    Y(mm)    Z(mm)  KinE(MeV)  dE(MeV)  StepLeng  TrackLeng  NextVolume  ProcName
  0      47.4      -53      -150       6         0         0         0      Envelope  initStep
  1      47.4      -53      -58      0.844      0         92        92      Envelope  compt
  2      -46       15.9      5.55      0.47      0        132       224      Envelope  compt
  3     -100       6.37     -3.62      0.47      0        55.5      280      World
Transportation
  4     -120       2.84     -7.02      0.47      0        20.6      301     OutOfWorld
Transportation
*****
* G4Track Information:  Particle = e-,   Track ID = 3,   Parent ID = 1
*****

Step#    X(mm)    Y(mm)    Z(mm)  KinE(MeV)  dE(MeV)  StepLeng  TrackLeng  NextVolume  ProcName
  0      -46       15.9      5.55      0.375      0         0         0      Envelope  initStep
  1     -46.1      16.4      5.98      0.0482     0.327     1.16      1.16     Envelope  eIoni
  2     -46.1      16.3      5.98      0         0.0482    0.0408     1.2     Envelope  eIoni
```

Primary  $\gamma$

Compton  $e^-$



# Geant4 production cuts

---

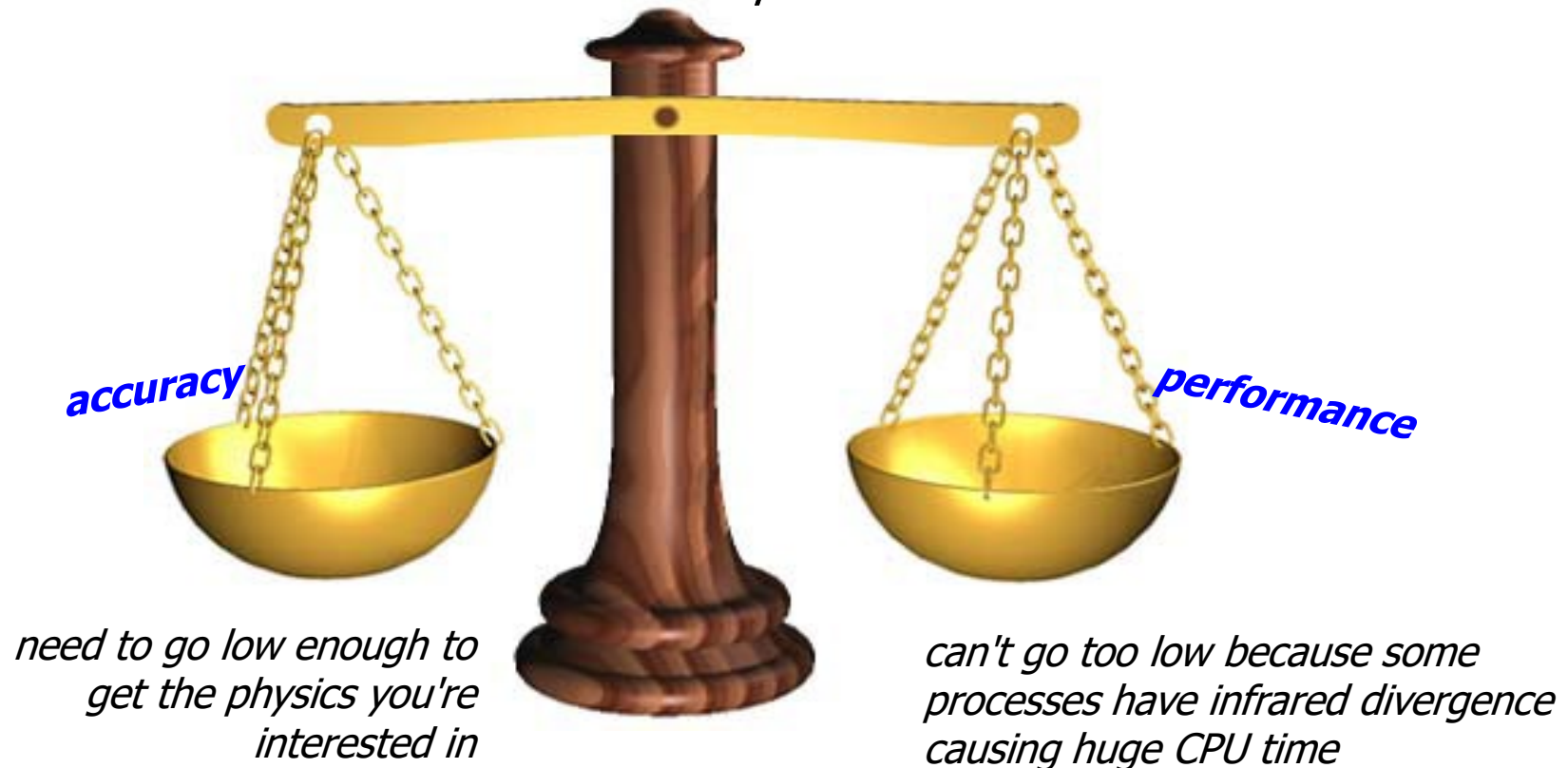
- Geant4 does not have tracking cuts
  - All tracks are followed **down to zero energy**
    - ..or until they leave the world volume or are destroyed in interactions
  - Could be implemented **manually** by the user
- Geant4 uses only a **production cut** → **"range production threshold"**
  - i.e. cuts deciding whether a **secondary** particle to be **produced** or not
    - AlongStep vs. PostStep
    - Applies **only** to:  $\gamma$  from **bremsstrahlung**,  $e^-$  from **ionization** and low-energy **protons** from **hadronic elastic scattering**
  - This threshold is a **distance**, not an energy
    - Particles unable to travel at least the range cut value are **not produced**
- One **production threshold** is uniformly set
  - Sets the **"spatial accuracy"** of the simulation
- Production threshold is **internally converted** to the **energy threshold**, depending on *particle* type and *material*



# Production cut

- Key ingredient of the mixed MC: **threshold**

*the best compromise*





# Cuts – UI commands

---

```
# Universal cut (whole world, all particles)
/run/setCut 10 mm

# Override low-energy limit
/cuts/setLowEdge 100 eV

# Set cut for a specific particle (whole world)
/run/setCutForAGivenParticle gamma 0.1 mm

# Set cut for a region (all particles)
/run/setCutForARegion myRegion 0.01 mm

# Print a summary of particles/regions/cuts
/run/dumpCouples
```



## Part III: Physics lists & Co.

---



# A physics list: what it is, what it does

---

- One instance per application
  - registered to run manager in `main()`
  - inheriting from `G4VUserPhysicsList`
- Responsibilities
  - all particle types (electron, proton, gamma, ...)
  - all processes (photoeffect, bremsstrahlung, ...)
  - all process parameters (...)
  - production cuts (e.g. 1 mm for electrons, ...)



# G4VUserPhysicsList

---

- All **physics lists** **must** derive from this class
  - And then be **registered** to the G4(MT)RunManager
  - **Mandatory** class in Geant4

```
class MyPhysicsList: public G4VUserPhysicsList {  
public:  
    MyPhysicsList();  
    ~MyPhysicsList();  
    void ConstructParticle();  
    void ConstructProcess();  
    void SetCuts();  
}
```

- User must implement the following (purely virtual) **methods**:
  - `ConstructParticle()`, `ConstructProcess()`
- **Optional Virtual method**:
  - `SetCuts()` (used to be purely virtual up to 10.2)

# Three ways to get a physics list



---

- **Manual:** Write your own class, to specify all particles & processes that may occur in the simulation (very flexible, but difficult)
- **Physics constructors:** Combine your physics from pre-defined sets of particles and processes. Still you define your own class – modular physics list (easier)
- **Reference physics lists:** Take one of the pre-defined physics lists. You don't create any class (easy)



# Derived class from G4VUserPhysicsList

- Implement 3 methods:

```
class MyPhysicsList : public G4VUserPhysicsList {  
public:  
    // ...  
    void ConstructParticle();    // pure virtual  
    void ConstructProcess();    // pure virtual  
    void SetCuts();  
    // ...  
}
```

**Advantage:** most flexible

**Disadvantages:**

- most verbose
- most difficult to get right



# G4VUserPhysicsList: implementation

---

- `ConstructParticle()`
  - choose the particles you need in your simulation, define **all of them** here
- `ConstructProcess()`
  - for each particle, assign **all the physics processes** relevant to your simulation
- `SetCuts()`
  - set the **range cuts for secondary production** for processes with infrared divergence





# G4VModularPhysicsList

- Similar structure as **G4VUserPhysicsList** (same methods to override – though not necessary):

```
class MyPhysicsList : public G4VModularPhysicsList {  
public:  
    MyPhysicsList();           // define physics constructors  
    void ConstructParticle();   // optional  
    void ConstructProcess();    // optional  
    void SetCuts();             // optional  
}
```

## Differences to “manual” way:

- Particles and processes typically handled by **physics constructors** (still customizable)
- **Transportation** automatically included



# Physics constructors (1)

---

- **"Building blocks"** of a modular physics list
- Inherit from **G4VPhysicsConstructor**
- Defines **ConstructParticle()** and **ConstructProcess()**
  - to be fully imported **in modular list** (behaving in the same way)
- **GetPhysicsType()**
  - enables **switching physics** of the same type, if possible (see next slide)



## Physics constructors (2)

---

- Huge set of **pre-defined ones**
  - **EM**: Standard, Livermore, Penelope
  - **Hadronic inelastic**: QGSP\_BIC, FTFP\_Bert, ...
  - **Hadronic elastic**: G4HadronElasticPhysics, ...
  - ... (decay, optical physics, EM extras, ...)
- You can implement **your own** *(of course)* by **inheriting** from the **G4VPhysicsConstructor** class

Code: `$G4INSTALL/source/physics_lists/constructors`



# How to use physics constructors

Add **physics constructor** in the class **constructor**:

```
MyModularList::MyModularList() {  
    // Hadronic physics  
    RegisterPhysics(new G4HadronElasticPhysics());  
    RegisterPhysics(new G4HadronPhysicsFTFP_BERT_TRV());  
    // EM physics  
    RegisterPhysics(new G4EmStandardPhysics());  
}
```

This **already works** and no further method overriding is necessary 😊



# Reference physics lists

---

- Pre-defined ("plug-and-play") physics lists
  - already containing a **complete set** of particles & processes (that work together)
  - **targeted** at specific area of interest (HEP, medical physics, ...)
  - constructed as **modular physics lists**, built on top of **physics constructors**
  - **customizable** (by calling appropriate methods before initialization)



# Using a reference physics list

- Super-easy: in the `main()` function, just register an instance of the physics list to the **G4 (MT) RunManager**:

```
#include "QGSP_BERT.hh"

int main() {
    // Run manager
    auto* runManager = G4RunManagerFactory::CreateRunManager();
    // ...
    G4VUserPhysicsList* physics = new QGSP_BERT();
    // Here, you can customize the "physics" object
    runManager->SetUserInitialization(physics);
    // ...
}
```

# The complete lists of Reference Physics List

`$G4INSTALL/source/physics_lists/lists`

FTF\_BIC.hh

FTFP\_BERT.hh

FTFP\_BERT\_HP.hh

FTFP\_BERT\_TRV.hh

FTFP\_INCLXX.hh

FTFP\_INCLXX\_HP.hh

G4GenericPhysicsList.hh

G4PhysListFactoryAlt.hh

G4PhysListFactory.hh

G4PhysListRegistry.hh

G4PhysListStamper.hh

INCLXXPhysicsListHelper.hh

LBE.hh

NuBeam.hh

QBBC.hh

QGS\_BIC.hh

QGSP\_BERT.hh

QGSP\_BERT\_HP.hh

QGSP\_BIC\_AllHP.hh

QGSP\_BIC.hh

QGSP\_BIC\_HP.hh

QGSP\_FTFP\_BERT.hh

QGSP\_INCLXX.hh

QGSP\_INCLXX\_HP.hh

Shielding.hh



[Docs](#) » Reference Physics Lists

## Reference Physics Lists

A detailed description of key reference physics lists which are included within the source tree of the GEANT4 toolkit. A an incomplete selection of diverse lists is described here in terms of the components within the list and possible use cases and application domains.

### Contents:

- [FTFP\\_BERT Physics List](#)
  - [Hadronic Component](#)

# Where to find information?



<https://www.geant4.org/docs/>

## Geant4 Documentation

This page gives you an overview of all available documents which are created and maintained by the Geant4 international collaboration.

### Introduction to Geant4

[html](#) - [pdf](#) - [epub](#) - [kindle](#)

This document gives you a more complete introduction to Geant4.

### Installation Guide

[html](#) - [pdf](#) - [epub](#) - [kindle](#)



#### On this page

[Introduction to Geant4](#)

[Installation Guide](#)

[User guides](#)

[For Application Developers](#)

[For Toolkit Developers](#)

[Physics Reference Manual](#)

[Physics List Guide](#)

[Examples](#)

[Frequently Asked Questions](#)

[Geant4 source code](#)



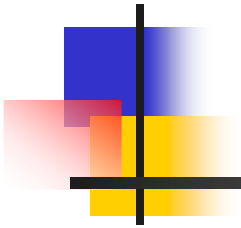


# Summary – three kinds of physics lists for Geant4

---

- Old-style flat physics list
  - You code **what you want**, particle by particle and process by process
  - Very much flexible, but **not really encouraged**
- User-custom modular physics list
  - **Blocks** (constructors) **provided** by Geant4
  - Can register **user-custom** constructors
  - Usually the *optimal compromise* between flexibility and user-friendliness
- Ready-for-the-use Geant4 physics list
  - **Plug and play** (directly registered in the main!)
  - Can still register **extra constructors**

# Part IV: Physics processes and models



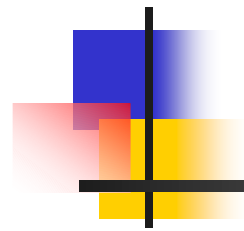


# Philosophy

---

- Provide a **general model framework** that allows the **implementation** of **complementary/alternative models** to **describe the same process** (e.g. Compton scattering)
  - A given **model** could work better in a certain **energy range**
- **Decouple** modeling of cross sections and of final state generation
- Provide **processes** containing
  - Many possible models and cross sections
  - Default cross sections for each model

**Models under continuous development**



# Electromagnetic physics

---

# Inventory (and specs) of the models for $\gamma$ -rays

1 MeV  $\gamma$  in Al

- Many models available for each process
  - Plus one full set of polarized models
- Differ for energy range, precision and CPU speed
  - Final state generators
- Different mixtures available the Geant4 EM constructors

Model	$E_{\min}$	$E_{\max}$	CPU
G4LivermoreRayleighModel	100 eV	10 PeV	1.2
G4PenelopeRayleighModel	100 eV	10 GeV	0.9
G4KleinNishinaCompton	100 eV	10 TeV	1.4
G4KleinNishinaModel	100 eV	10 TeV	1.9
G4LivermoreComptonModel	100 eV	10 TeV	2.8
G4PenelopeComptonModel	10 keV	10 GeV	3.6
G4LowEPComptonModel	100 eV	20 MeV	3.9
G4BetheHeitlerModel	1.02 MeV	100 GeV	2.0
G4PairProductionRelModel	10 MeV	10 PeV	1.9
G4LivermoreGammaConversionModel	1.02 MeV	100 GeV	2.1
G4PenelopeGammaConversionModel	1.02 MeV	10 GeV	2.2
G4PEEFfluModel	1 keV	10 PeV	1
G4LivermorePhotoElectricModel	10 eV	10 PeV	1.1
G4PenelopePhotoElectricModel	10 eV	10 GeV	2.9

Similar situation for  $e^{\pm}$



# EM concept

---

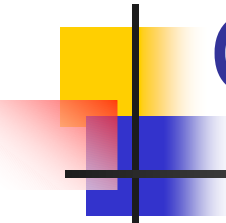
- The **same physics processes** (e.g. Compton scattering) can be described by **different models**, that can be **alternative** or **complementary** in a given energy range
- For instance: **Compton scattering** can be described by
  - **G4KleinNishinaCompton**
  - **G4LivermoreComptonModel** (specialized low-energy, based on the Livermore database)
  - **G4PenelopeComptonModel** (specialized low-energy, based on the Penelope analytical model)
  - **G4LivermorePolarizedComptonModel** (specialized low-energy, Livermore database with polarization)
  - **G4PolarizedComptonModel** (Klein-Nishina with polarization)
  - **G4LowEPComptonModel** (full relativistic 3D simulation)
- Different models can be **combined**, so that the appropriate one is used in each given energy range (→ performance optimization)


# When/why to use Low Energy Models



- **Use** Low-Energy models (Livermore or Penelope), as an *alternative* to Standard models, when you:
  - need **precise treatment** of EM showers and interactions at **low-energy** (keV scale)
  - are interested in **atomic effects**, as fluorescence x-rays, Doppler broadening, etc.
  - can afford a more **CPU-intensive** simulation
  - want to **cross-check** an other simulation (e.g. with a different model)
- **Do not use** when you are interested in EM physics **> MeV**
  - same results as Standard EM models, **performance penalty**

# EM Physics Constructors for Geant4 10.4 - ready-for-the-use



G4EmStandardPhysics	– default
G4EmStandardPhysics_option1	– HEP fast but not precise
G4EmStandardPhysics_option2	– Experimental
G4EmStandardPhysics_option3	– medical, space
G4EmStandardPhysics_option4	– optimal mixture for precision
G4EmLivermorePhysics	 <div>Combined Physics Standard &gt; 1 GeV <b>LowEnergy &lt; 1 GeV</b></div>
G4EmLivermorePolarizedPhysics	
G4EmPenelopePhysics	
G4EmLowEPPhysics	
G4EmDNAPhysics_option...	

...

- Advantage of using of these classes – they are **tested on regular basis** and are used for regular validation





# Hadronic physics

---

(a very quick overview)



# Hadronic Physics

---

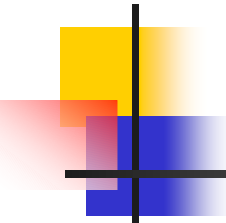
- Data-driven models
- Parametrised models
- Theory-driven models



# Hadronic physics challenge

---

- Three energy regimes
  - $< 100$  MeV
  - resonance and cascade region (100 MeV - 10 GeV)
  - $> 20$  GeV (QCD strings)
- Within each regime there are several models
- Many of these are phenomenological



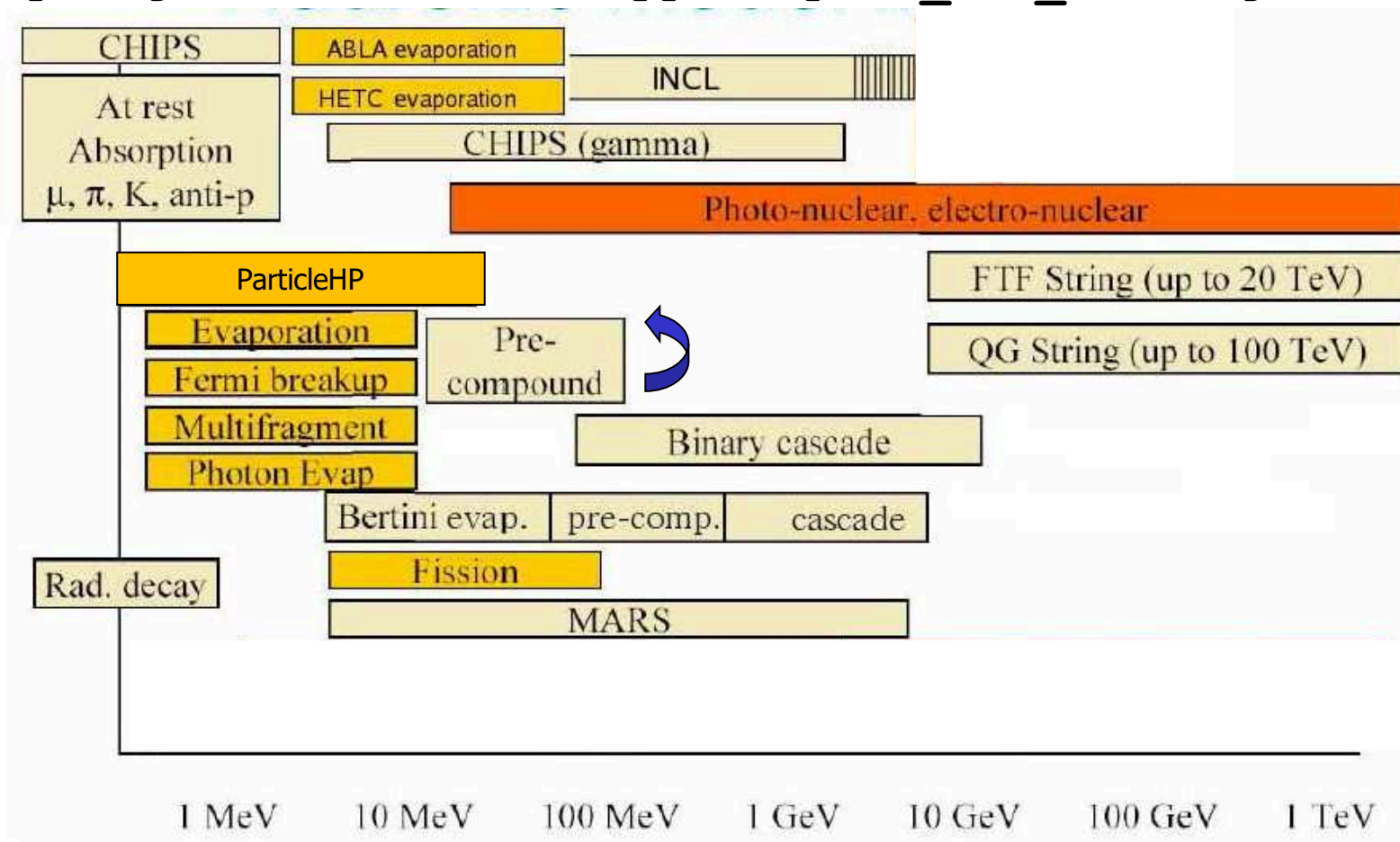
# Reference physics lists for Hadronic interactions

---

- **Two families** of builders for the high-energy part (p, n,  $\bar{n}$  and K)
  - **QGS**, or list based on a model that use **the Quark Gluon String model** for high energy hadronic interactions
  - **FTF**, based on the FTF (FRITIOF like string model)
- **Three families** for the **cascade** energy range
  - **BIC**, binary cascade
  - **BERT**, Bertini cascade
  - **INCLXX**, Liege Intranuclear cascade model
- "High precision" (**HP**) option, below 20 MeV
  - Database tracking for **n, p, d, t,  $^3\text{He}$  and  $\alpha$**
  - Data from ENDFVII.r1 or TENDL-2014
  - CPU-thirsty

# Hadronic model inventory

[http://geant4.cern.ch/support/proc\\_mod\\_catalog/models](http://geant4.cern.ch/support/proc_mod_catalog/models)





# Hands-on session

---

- Task3
  - Task3a: Particles and processes
  - Task3b: Physics lists
  - Task3c: Production cuts
- <http://geant4.lns.infn.it/alghero2023/task3>



# Bonus: FastSimInterface

---

- *In main:*

```
G4FastSimulationPhysics* fastSimulationPhysics = new G4FastSimulationPhysics();
fastSimulationPhysics->BeVerbose();
// -- activation of fast simulation for particles having fast simulation models
// -- attached in the mass geometry:
fastSimulationPhysics->ActivateFastSimulation("e-");
fastSimulationPhysics->ActivateFastSimulation("e+");
// -- Attach the fast simulation physics constructor to the physics list:
physicsList->RegisterPhysics( fastSimulationPhysics );
```

- **Fast simulation** completely **stops** the **standard Geant4 processes** at the step of Fast Simulation model and then resumes them.
- Is activated **only** in a **certain G4Region** at a **certain condition** and only for **certain particles**.
- **Easiest** way to implement external code into Geant4.

# Where: in a *G4Region* defined in *DetectorConstruction*

## ● Add to *DetectorConstruction::Construct()*

```
//my volume  
G4Box* MySolid = new G4Box("MyBox",SizeX/2,SizeY/2,SizeZ/2.);  
G4LogicalVolume* MyVolumeLogic = new G4LogicalVolume(MySolid,MyMaterial,"MyVolume");  
new G4PVPlacement(Rotation,Position,MyVolumeLogic,"MyVolume",logicWorld,false,0);
```

**Volume declaration  
(completely standard)**

```
//my region (necessary for the FastSim model)  
fRegion = new G4Region("MyRegion");  
fRegion->AddRootLogicalVolume(MyVolumeLogic);
```

**G4Region declaration**

**Add Logic Volume to  
G4Region declaration**

## ● Add to *DetectorConstruction::ConstructSDandField()*

```
void DetectorConstruction::ConstructSDandField()  
{  
    // ----- fast simulation -----  
    //extract the region of the crystal from the store  
    G4RegionStore* regionStore = G4RegionStore::GetInstance();  
    G4Region* MyRegion = regionStore->GetRegion("MyRegion");  
  
    //create the channeling model for this region  
    MyFastSimModel* MyModel = new MyFastSimModel("ChannelingModel",MyRegion);  
  
    //some options of the model...  
}
```

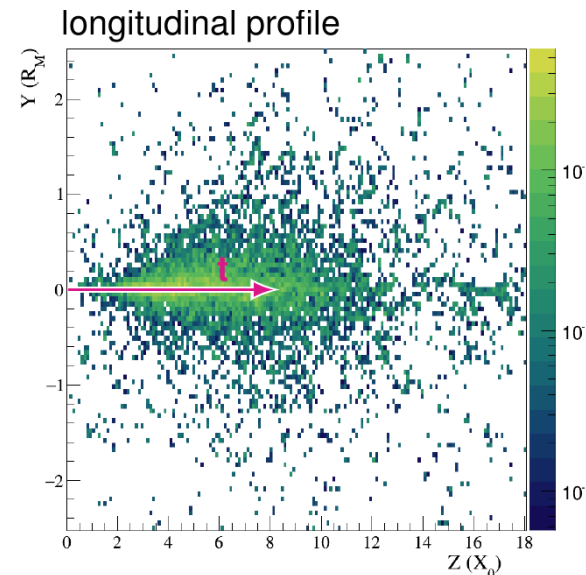
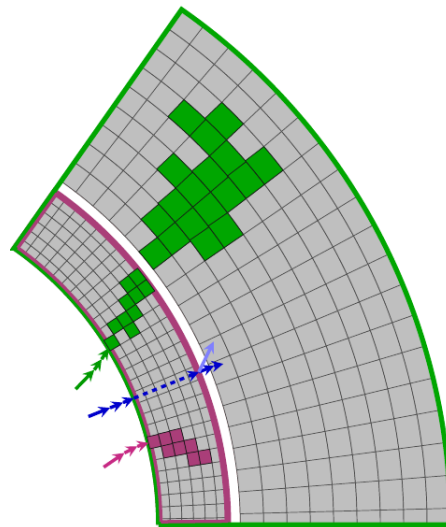
**Get G4Region**

**Declare your FastSim model**



# FastSimInterface: Applications

- Simulation of **electromagnetic showers in matter (e.m. calorimeters, ...)**
- Simulation of **sampling calorimeters**
- **Machine Learning**
- Implementation of **external codes** into Geant4



*From Anna Zaborowska presentation,  
examples/extended/parameterisations/Par01*